N91-20679

EXPERT SYSTEM DECISION SUPPORT FOR LOW-COST LAUNCH VEHICLE OPERATIONS

Dr. G.P. Szatkowski and Barry E. Levin

GENERAL DYNAMICS
Space Systems Division

5001 Kearny Villa Road
San Diego, CA 92138
(619) 496-7093

Sponsored by: USAF/Air Force Space Division

ABSTRACT

This describes progress in assessing the feasibility, benefits, and risks associated with AI Expert Systems applied to low cost expendable launch vehicle systems. This work was funded under the joint USAF/NASA Advanced Launch System (ALS) Program as applied research. Part One identified potential application areas in vehicle operations and on-board functions, assessed measures of cost benefit, and finally identified key technologies to aid in the implementation of decision support systems in this environment. Part Two of the program began the development of prototypes to demonstrate real-time vehicle checkout with controller and diagnostic/ analysis intelligent systems, and to gather true measures of cost savings vs. conventional software, verification and validation (V&V) requirements, and maintainability improvement.

The Expert System advanced development projects (ADP-2301 & 2302) main objective was to provide robust intelligent system for control/analysis that must be performed within a specified real-time window, in order to meet the demands of the given application. Timing is defined as responding to new data frames of 0.1 to 1.0 second intervals. Here we describe our efforts to develop two prototypes. Prime emphasis was on a controller expert system to show real-time performance in a cryogenic propellant loading application, and safety validation implementation of this system experimentally at the USAF Cape Canaveral Air Force Station Atlas Complex-36, using commertial-off-the-shelf software (cots) tools and object oriented programming (oop) techniques. This 'Smart GSE' (Ground Support Equipment) prototype is based in C, with imbedded expert system rules written in the CLIPS protocol. The relational database ORACLE® provides non-real-time data support.

The second demonstration develops the Vehicle/Ground Intelligent Automation concept, from Phase-I, to show cooperation between multiple expert systems (and conventional software modules). This 'Automated Test Conductor' (ATC) prototype utilizes a Knowledge-bus approach for intelligent information processing by use of virtual sensors and blackboards to solve complex problems. It incorporates distributed processing of real-time data and object-oriented techniques for command, configuration control, and auto-code generation.

BACKGROUND

The Air Force and NASA have recognized that our nation's current suite of launch vehicle systems has a number of problems making them inadequate for the projected needs after the late-1990's. A reduction in cost to $300/lb of payload delivery, to LEO, 0.999+ reliability, high resiliency (elimination of long standdowns of many months), and high launch rate capacity are reasons behind the joint USAF/NASA effort for an operational ALS and Shuttle 'C'. ALS will serve the commercial and DoD mission models beginning in 2000. In order to meet the goals of $300/lb and launch rates as high as 25 missions annually, on-board systems and their associated ground operations segment must be made as autonomous as possible, while at the same time improving reliability and safety. Under the ALS Program, a study was initiated to explore the use of EXPERT knowledge-based system (KBS) techniques for the purpose of automating the decision processes of these vehicles and all phases of the ground operations segment by assessing the feasibility, benefits, and risks involved.

An expert decision aid is a software approach to solving particular problems that are constantly changing over time and are complex or adaptive in behavior, the opposite of an analytical problem that is basically deterministic. Examples of these types of problems are: the re-scheduling of a vehicle checkout due to a damaged cable; or, determining if a system is indeed faulty given conflicting sensor readings. These heuristic problems require a depth of knowledge and experience (art rather than science) to form solutions quickly. Expert systems embody that collection of knowledge and experience in modular pieces that are rules and facts that describe the proper thought process for a given set of circumstances arrived at by any path. It is this modular independence that makes expert systems attractive. The incremental improvement of knowledge and experience can be built and tested readily without re-testing the rest of the software system, unlike conventional software that is difficult to maintain in a day to day changing environment [1].

PROJECT DESCRIPTION

The objective of this program is to develop, demonstrate, and evaluate the use of expert decision aids in areas that would improve ground and on-board system autonomy for the purpose of reducing the life cycle costs, shortening the processing critical path time, and improving safety and vehicle reliability. This technology program continued the work begun under the Phase-I study: Space Transportation Expert System Study (STRESS), contract managed by the USAF Wright Research and Development Center [2].

Tasks consists of an assessment of cost benefits vs implementation risks for specific applications, and demonstrations of key performance requirements to show feasibility within the selected application environment. Experience from our launch vehicle programs and other R&D

efforts shows that there are many opportunities in operations that reduce costs and improve autonomy, including:

- Ground operations: daily planning support and timely work-around decisions aids
- Ground checkout: autonomous operations and control
- On-board systems: monitoring, integration, and control
- Launch day: fly-with-fault diagnostics and decision aids

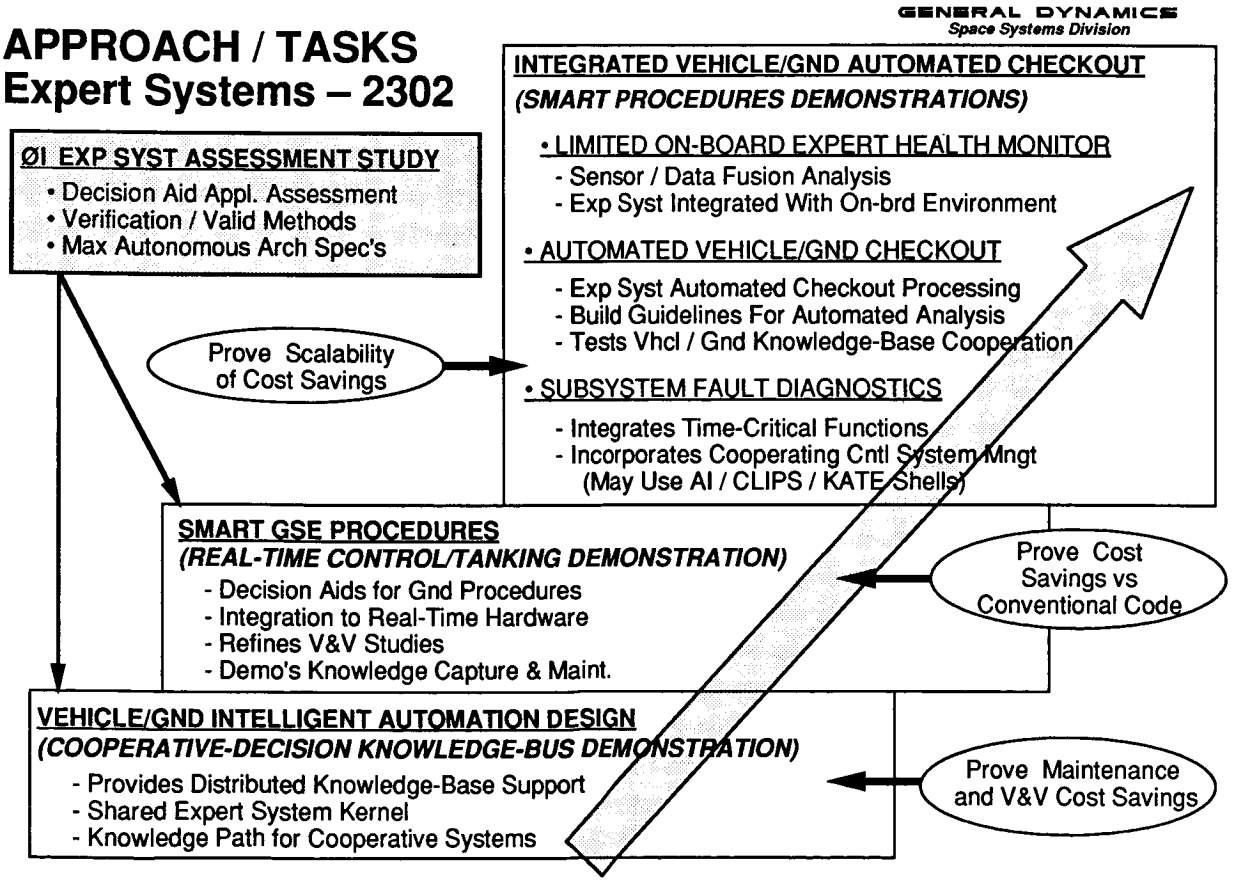
From 33 applications identified in Phase-I [3], cryogenic propellant tank loading was selected for a performance feasibility demonstration in order to reduce the risk of commitment to this developing technology. The Smart GSE prototype was of a fractional scale, sufficient to give a good performance correlation to a full-scale implementation. This demonstration intended to show:

- Ease of human interface to facilitate maintainability at the non-technical level;
- Real-time system performance for an appropriate level of complexity;
- Integration to both vehicle and ground hardware, and data systems;
- Validation methodology consistent for ground and on-board applications [4].

The second objective, targets the incorporation of KBSs into a combined Vehicle / Ground Intelligent Automation System. The "Knowledge-Bus" (K-bus) architecture, conceived in Phase-I, was used as the baseline concept in developing a maintainable mix of multiple conventional and knowledge-based systems. This layered architecture supports modularity and reusability of KBS components via object oriented programming techniques (hierarchial components, inheritance of methods, and polymorphism of functions) and knowledge-base partitioning [3] (for R/T performance and V&V efficiency). A second prototype demonstration of an Automated Test Conductor (ATC) system was begun to show cooperating intelligent systems in operation using blackboards and virtual sensors (elements of the K-bus concept). This approach is being successfully used to support the NASA Space Station Program. Elements of this architecture are already available in commercial-off-the-shelf software products. Development of an overall integrated architecture early in the investigation provides a context and focus for future demonstration prototypes, and assure the synergy of their gains in both development and use in vehicle operations, for example Integrated Health Monitoring (IHM).

This technical approach is shown in Figure—1.

SMART-GSE Procedures — This development task focused on supporting real-time control of time critical vehicle operations using KBSs. The application selection from the 33 candidates, the ranking shown in Figure—2, was a difficult choice. The demonstration had to require actual real-time servicing of decisions. It had to be sufficiently complex in scale to provide a good source of data for performance issues and for costs. It had to be non-trivial, i.e. dealing with real data so as to provide detailed feedback for correctness to serve as a benchmark for verification/validation testing. The final choice was cryogenic propellant loading of the Atlas launch vehicle. Although this application ranked in the middle, it satisfied all the requirements. The prototype would:



Figure—1, Expert Systems ADP Approach

	CANDIDATE SYSTEM	Final Assessment
22	Mission Planning with Automated Navig	38 49
15	Pre-Flight Test Analysis	32 00
26	System Wide Event Correlation	30 38
28	Facilities Manager	29 34
29	Mission Design Automation	29 27
16	Post Flight Telemetry Data Analysis	28 72
27	Vehicle Test Conductor/Scheduler	27 15
24	Command and Control Scheduler	26 78
31	Payload Manifesting	23 81
11	Vehicle Processing Logger System	23 05
10	Operation Troubleshooting	22 15
9	Launch Complex Environ Control System	19 48
6	Integrated Test Ctlr for Vehicle System	19 27
5	Operator Training Simulator	19 14
19	Propellant Tanking of Vehicle	18 13
18	Pneumatics, Press., and Purge Controls	17 03
25	Support for the Decision to Launch	16 79
32	Countdown Operations System Monitor	16 77
23	Range Safety System	16 32
8	Automatic Recorder Assignment	16 05
3	Critical Parameter Vehicle Surveillance	15 61
21	Guidance Calibration	15 25
2	Limit Testing	14 85
33	Telemetry/Landlines Checks & Assignm	14 57
17	Flight Control Power Applic and Monitor	13 78
30	Range Safety Sys. and Recovery Operatic	13 55
7	Automatic Remote Sensor Calibration	13 37
4	Hazardous Gas Identification and Subsy	12 61
20	Engine Ignition Ground Perform Mon	11 75
12	In Flight Engine Perform Monitor	8 03
13	Fluids Analysis Health Monitoring	7 83
14	Abort/Alternative Mission Modes (AGN&	6 41
1	Data Compression Analysis	5 58

Figure—2, Final Relative AI Candidate Ranking

- Prove R/T integration with GSE hardware and software by being installed at USAF Cape Canaveral Air Force Station Complex (CX)-36. Timing being defined in terms of 1 second decision loops based on monitored feedback
- Provide the V&V benchmark by using the validated Tanking Simulator available in the laboratory and later by using the pad validation equipment
- Provide comparative development cost data wrt the same task being done in conventional S/W on Titan/Centaur
- Provide long term maintenance cost data by being put into service at CX-36 and compared to Titan/Centaur

The basic approach to the Smart GSE system was a tanking controller built on a workstation using tools and standards so as to make it portable to any variety of computer systems. The demo used the CLIPS expert system shell from NASA/JSC. This shell had shown promise in our internal R&D efforts as capable of supporting real-time operations with suitable extensions.

We based all graphical interfaces on the X-window standard and Object Oriented Programming techniques. Here we used the Transportable Application Environment (TAE) provided by NASA/Goddard. This oop tool works on nearly all workstations and Macintosh systems under X. To provide the R/T feedback into the controller expert system, we used a PC version of our existing validated Tanking Simulator connected via a RS232 interface to the SUN workstation. We were using a SUN 3 system but planned to move the demo to a SUN 4 to do the timing tests. Later this would be moved to a Silicon Graphics workstation system for porting to CX-36.

Automated Test Conductor (ATC) — This development task focused on supporting the integration of KBSs into the vehicle processing environment; i.e. to actually have multiple expert systems cooperate in the performance of a given operation. The approach was to use the K-bus techniques in a judicious fashion to demonstrate the concept was do-able and without costly overhead making the concept impractical. The demonstration would prove that it is possible to have distributed knowledge-base support for control activities. This would open the door for accepting many of the 33 applications detailed in the Phase-I report. The driving force is the potential cost savings by having shared software kernels, object encapsulation of practices, procedures, and knowledge — to reduce validation, maintenance, and training. Further, the automation can now extend into the management of systems and not just isolated operations.

The basic approach in this task was to use multiple expert system modules, orchestrated by an ATC module, and running concurrently on 3 or more workstations. The workstations are initially networked via Ethernet TCP/IP protocol using socket transfer. For the R/T control management demonstration, we are in the process of installing VME hardware linkages between the workstations for message passing. The software being developed followed the principles of the K-bus. Throughout this project we used standard 'C' language and the GNU 'C++' oop language. For UNIS (Unified Network Information System) interfacing we used our own SQL-based data bridge to the ORACLE® relational data management system. This provided specification data to the expert systems upon demand. The initial test was a simple cooperative tanking task between subsystems. Later the Smart GSE controller becomes integrated in this encompassing system.

SMART-GSE DEMONSTRATION

The system level requirements were based on inputs from the Atlas CX-36 design and the system architecture developed under the ALS basic pre-design effort. Figures—3,4 show the required tasks and data-flows at the level of the Propellant Tanking Manager and one of the 7 subsystems. The primary features are first, the separation of management/control from health monitoring for a R/T system; and second the separation of R/T feedback monitoring from diagnosis.

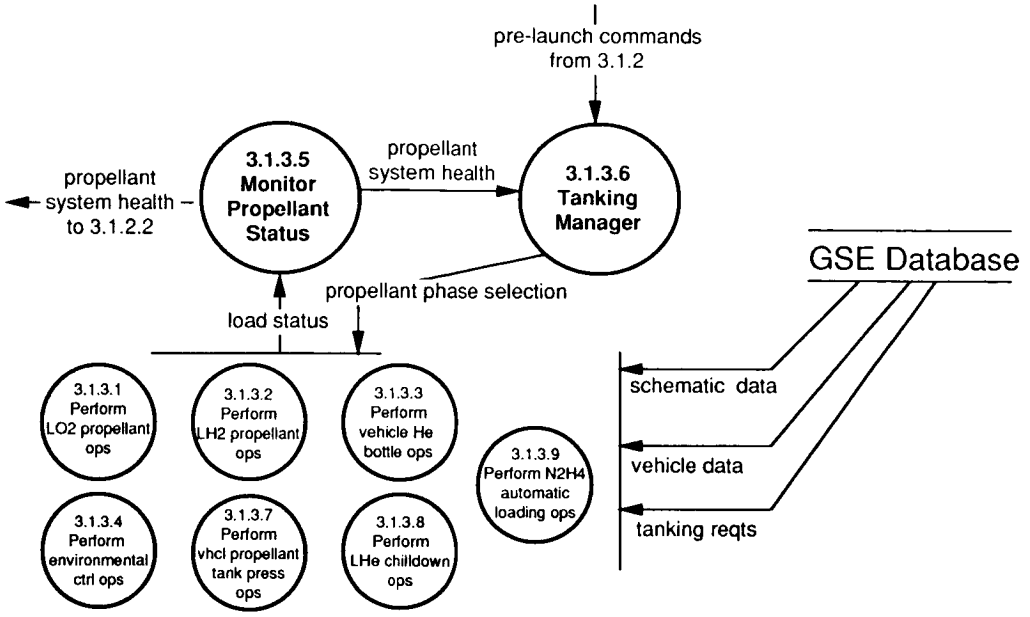
The prototype demonstration configuration is a laboratory closed-loop test linking the Tanking Simulator with the expert system controller. The Tanking Simulator consists of the existing Atlas GSE models with inputs from Ground Skid models and sequencing information from actual telemetry flight-test data. The Tanking Simulator is operated in a personal computer with a 100 millisecond cycle basis. This is connected via an RS232 line to a SUN 3 workstation running the expert system prototype. Preliminary validation tests were to have been done here. Later this system configuration would be modified to connect to the developmental launch control computers in San Diego for initial integration of the actual data telemetry streams. With this accomplished the transfer to CX-36 should be relatively straight forward. Testing at CX-36 would center on reading the telemetry data and determining performance variations in a variety of stressed situations. This testing would use the pad's validation equipment and not an actual vehicle. Timing test would be performed under a R/T Unix system on a Silicon Graphics workstation. Both the conventional equipment and the expert system would operate in parallel and comparisons of performance would be evaluated manually. The demonstration process is depicted in Figure—5.

The Smart GSE software configuration is shown in Figure—6 and highlights the significant elements. Some of these include the NASA/JSC: CLIPS expert system shell, the NASA/Goddard: TAE object oriented shell, and the ORACLE ® Relational Data Base System that was bridged to for specification type data as needed.

The primary emphasis in this project has been to demonstrate that expert systems can operate in real-time environments. It is

SMART GSE REQUIREMENTS (Level 3.1.3)
Expert Systems — ADP 2302

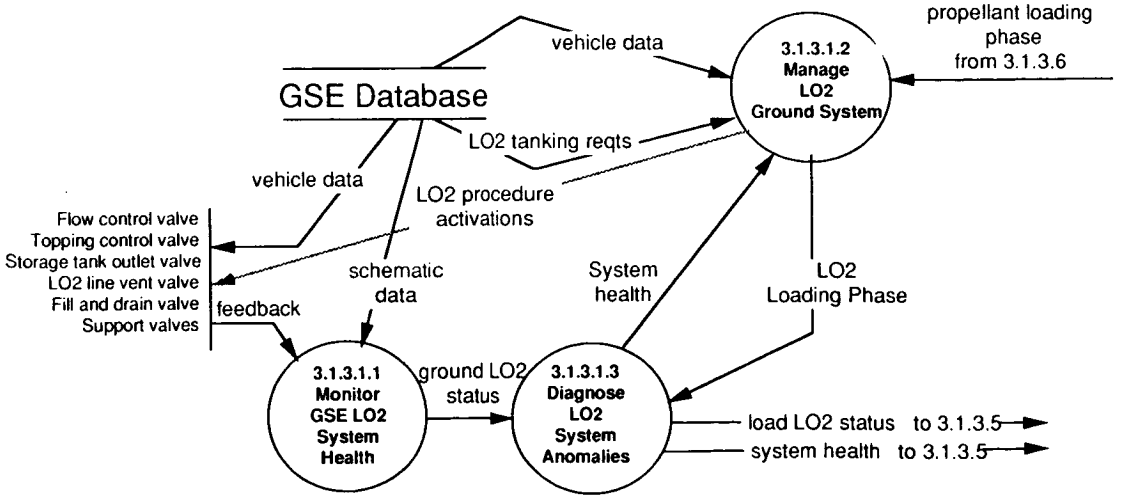
- Evaluated R/T Issues for Command & Cntl wrt ES
- Separated Mngt from Health Monitoring



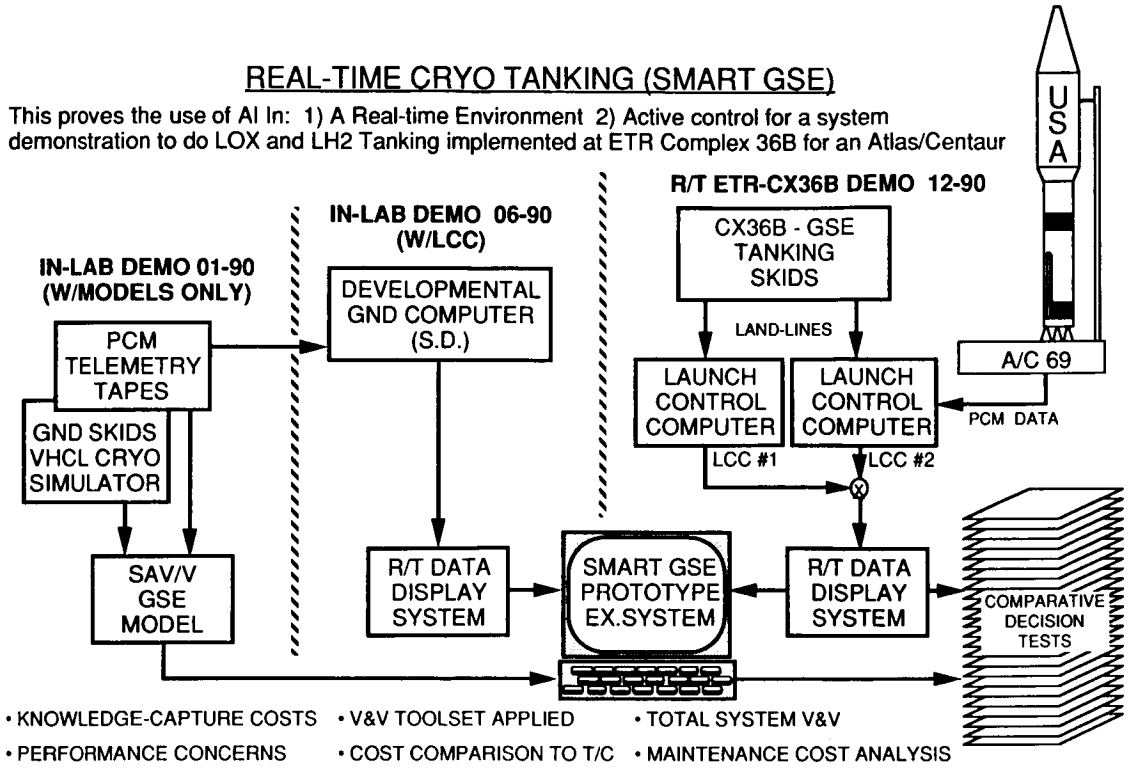
Figure—3, Propellant Manager requirements

SMART GSE RQTS - LO2 (Level 3.1.3.1)
Expert Systems — ADP 2302

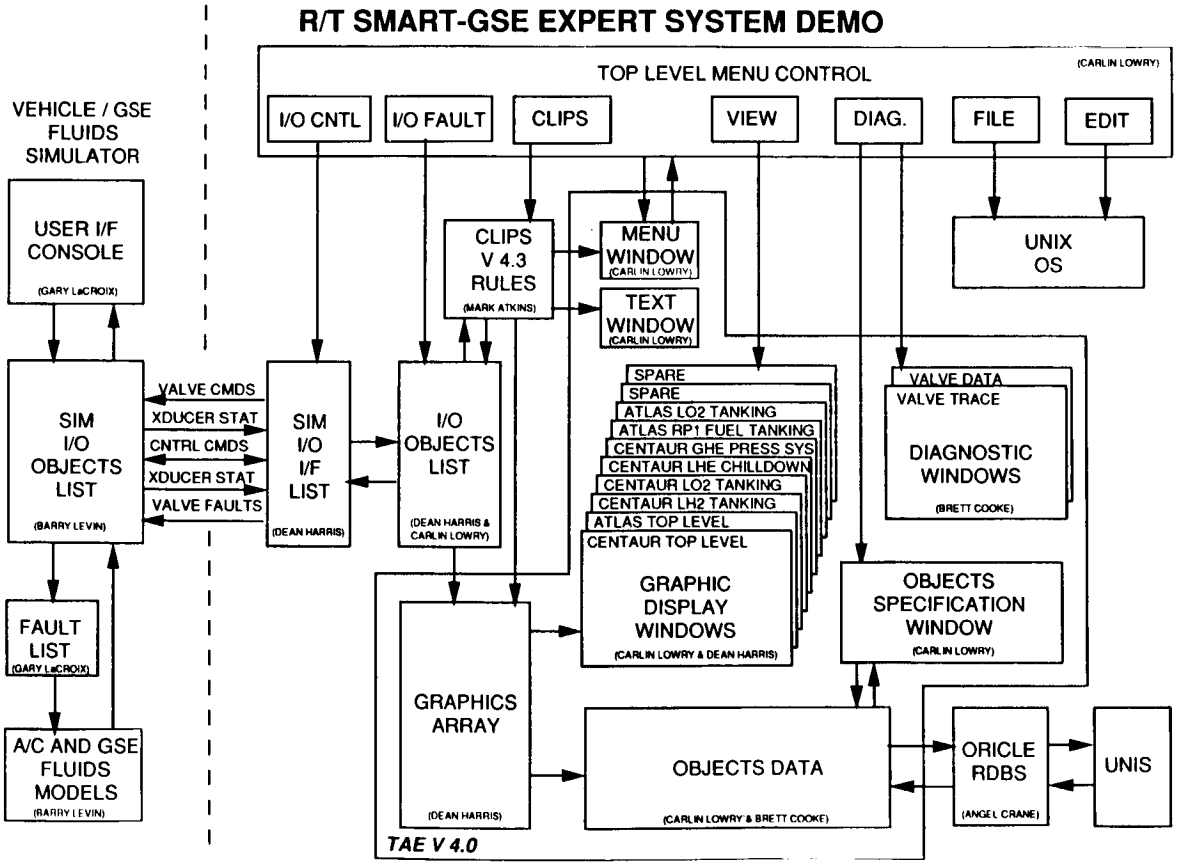
- Evaluated Knowledge-bus Issues wrt Command & Cntl
- Separated Monitoring from Diagnostics



Figure—4, LO2 Subsystem Manager requirements



Figure—5, Smart GSE shows evolution to full validation



Figure—6, Smart GSE prototype features include CLIPS, ORACLE, and TAE cots tools

anticipated that the CLIPS shell would not be the only expert system shell used in the course of the Project development. Aspects of the Vehicle/Ground Intelligent Automation study required for test planing and procedure coordination may better use cots expert systems shells for goal oriented activities.

The prototype Smart GSE demonstration actually consists of several expert systems working together. These are presently modules linked within the same program, as apposed to being true cooperating distinct systems.

Propellant Manager Expert System — This ES is required to accept a set of primary commands from the Mission Manager and perform the necessary operations on the hardware systems to accomplish the functions that the he has selected. A sample list of the commands that the operator can select is:

TANK, DE-TANK, START, STOP, LAUNCH, HOLD, STATUS, ...

The Propellant Manager ES portion of the Smart GSE then performs any necessary correlation between the different systems under its control and issues a set of secondary commands that will insure that the safety of the vehicle, pad, and launch area will be maintained. Further, the Propellant Manager ES coordinates the data about each of the subsystems and adjusts each to ensure that they function smoothly between systems and prelaunch phases. The data or status at this level is also sent up to the Mission Manager for evaluation.

Subsystem Controller Expert Systems — The Propellant Manager coordinates commands for several Subsystem Controllers. These sub expert system Controllers are for each of the different fluid systems (LH2, LO2, LHe, Hydrozine, ...) and act upon the commands issued by performing the functions that have been requested.

These Controllers are concerned with the detailed functional commands to control the physical hardware and turn on or off specific sequences of valves or other equipment. Typical command decision-sequences made by the local Controller on the functional elements would be:

FAST FILL, TOP OFF, PRESSURIZE, DUMP, PURGE ...

The subsystem Controllers process function (object) commands by accessing a call to a set of conversion routines that would take the function (object) request and perform the necessary hardware interpellation. This is actually turn on/off the necessary valves or equipment to perform the function in question. An example sequence of valves would be:

(F-34-5 ON) (F-35-5 ON) (F-7-1 OFF) (F-22 OFF) ...

These Controllers also have responsibility for selecting the primary or backup functions as necessary.

Health Monitoring Expert System — A Health Monitoring expert system correlates feedback information and statuses the health of the equipment. All critical function feedback is sent to both this Health Monitoring expert system and the expert system Controller. This direct feedback allows fast close-loop response by the Controller. Non-critical anomalies are analyzed by the Health Monitor expert system. These problems are evaluated using the expert system data fusion capability to ascertain if a true failure has occurred. It then isolates the cause of the failure and notifies the expert system Controller of the defect. This allows the Controller to select appropriate backup functional equipment. In parallel, the Health Monitor expert system notifies the Propellant Manager Health Monitor of the defect and requests repairs. All anomalies are reported to this top level Propellant Manager Health Monitor to correlate with the other active systems.

Simulation training and validation testing — A monitoring / debug mode of operation will be available on the controller side of the sub level expert system to make suggestions of what operation the operator should be selecting and why. In the validation phase a 'scripting' capability would be used for dynamic system testing.

Progress to Date — The controller begins with a simplified schematic display of all nine propellant systems. A menu bar resident at all times allows the human controller to choose options from pull-down menus. These features include viewing alternate schematics, performing diagnostics on the imbedded expert systems, and manipulating the program parameters to explore or control specific scenarios. A more detailed view of any subsystem replaces a top level schematic when the pertinent name field is clicked on the top level schematic itself or when the name is chosen from a pulldown menu.

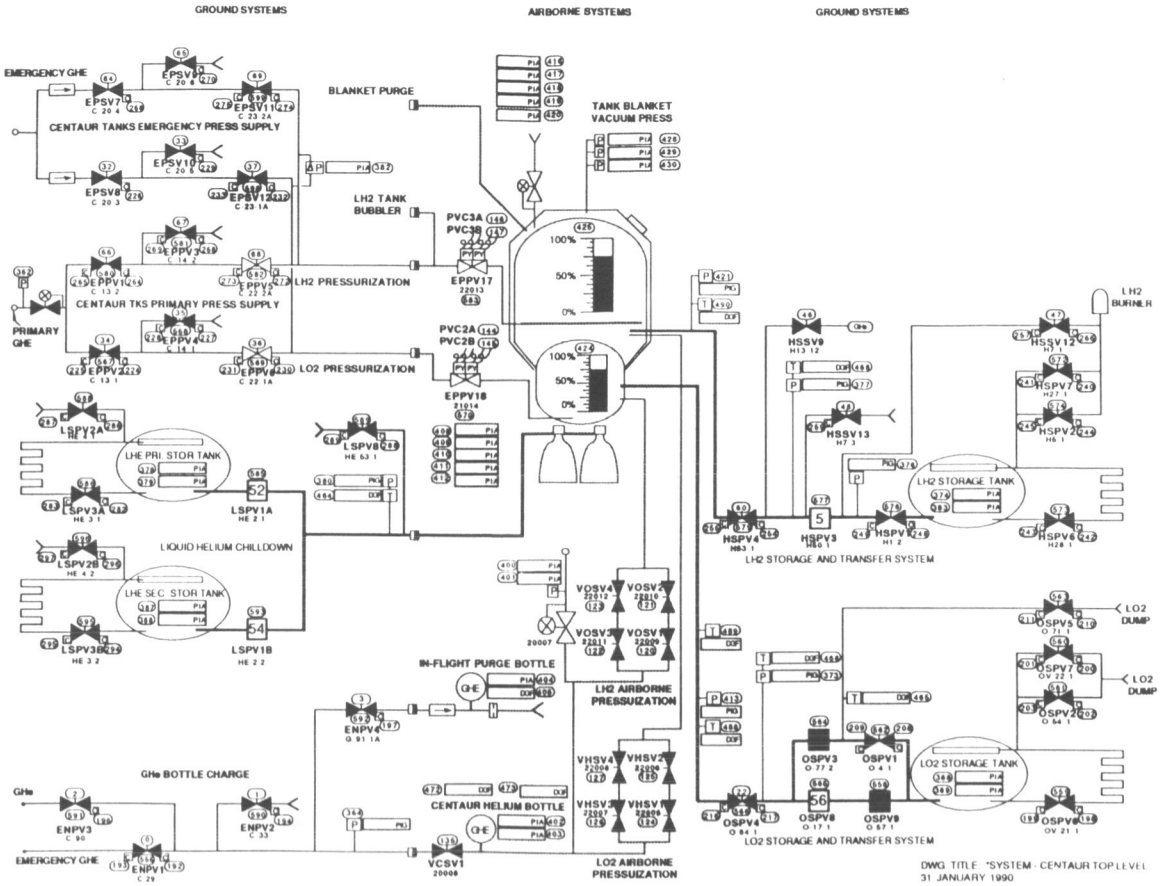
Each schematic consists of a background with overlaid discrete items. Valves, connections, and the tanks are discrete objects. The 'valve objects' change color to indicate changed state, such as open, closed, fault, or warning. A window containing both static and dynamic information about a schematic item is activated by clicking on the desired item. This information comes thru. the ORACLE® database bridge. A stretcher object reacts to the changes in the propellant flow and indicates the current tanking percentage completion on a sketch of the 'tank object'.

Figures—7 is the top functional level schematic; Figure—8 is a photograph from the SUN 3 display of the Smart GSE prototype. The photograph displayed is one of two top-level system displays, the one for the Centaur is shown and there is one for the Atlas vehicle/ground propellant systems. The remainding displays are sublayer displays of subsystems and show increased detail of all the operational components that facilitate the primary functions depicted on the top-level displays.

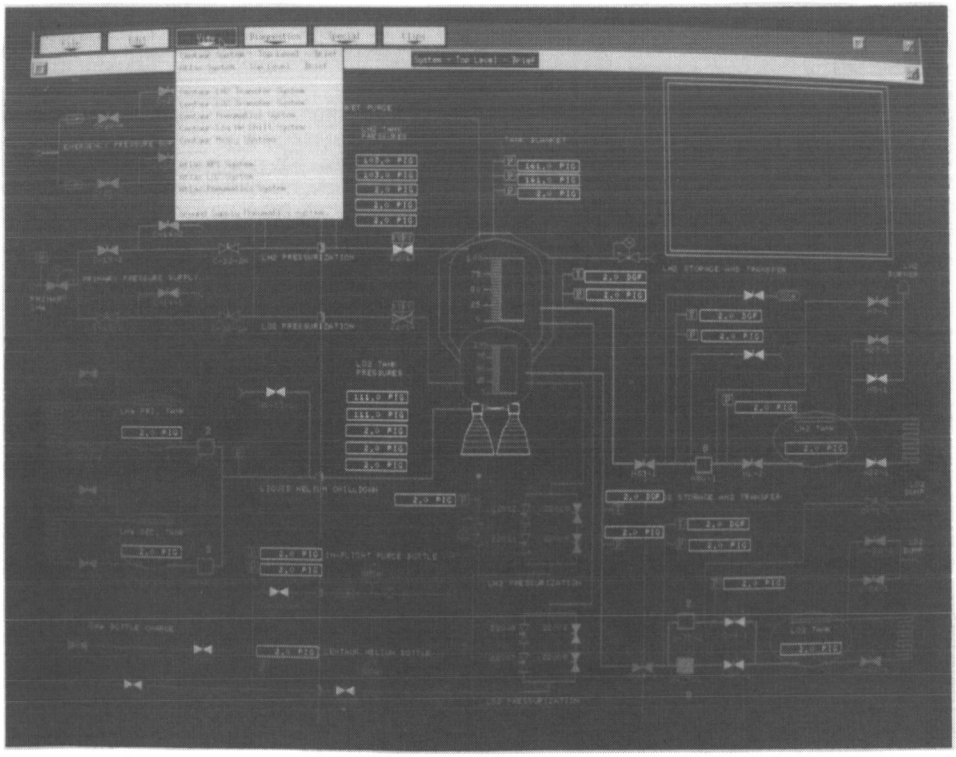
Operationally the ES knowledgebases for the LO2 and LH2 subsystems were completed and had begun initial integration testing. The displays were completed and had begun integration testing with the Tanking Simulator. The displays had not completed the animation software to show the actual closed-loop data feedback from the Tanking Simulator. The other subsystems knowledgebases were in various stages of development when work was terminated.

Real-Time Issues — The application as a controller implies control of the real-time decision process in relationship to the external environment that it deals with. Several methods were explored in an attempt to devise techniques for controlling the inferencing process and the rule-set that it evaluated. Some of these methods are shown in Figure—9. The use of SALIENCE is one of those techniques that is well suited to this area of expert systems. Simply put, SALIENCE is another term for rule-priority. How that priority is established, how the priority is maintained over time, and how the priority scheme interacts with the Inference Engine tie-breaking mechanism are all important application considerations [5].

The examination of rules that do not fire during an expert system application's cycle is the unfortunate overhead that expert systems typically carry. Another method used was to create a rule partitioning approach that would not require modification of the standard CLIPS shell (as tried in the Portable Inference Engine [6]), but would instead be application-specific CLIPS code that could reduce the number of rules present in the Rete Net at any given time. There are only a few expert system development tools that implement the concept of knowledge base rule 'clustering' where production rules are organized into logical arrangements to facilitate better control over their execution. An example of this approach



Figure—7, Top functional level SCHEMATIC for Centaur propellant systems



Figure—8, Top functional level OBJECT-ORIENTED DISPLAY

(called 'Function Control Blocks') is found in IBM's Expert System Environment (ESE). The challenge was to create a similar rule partitioning approach in CLIPS without any significant degradation in performance - i.e. continual real-time operation. The method [1] removes rule 'clusters' that were not used during a real-time cycle and adds any 'clusters' that are required during that same cycle. In developing our CLIPS approach, minimum overhead costs were unavoidable. To date, R/T operational performance has not been tested.

Using this approach, verification and validation techniques for expert systems are more likely to succeed than with traditional expert systems. By efficiently 'clustering' the rules and facts, modular testing of modifications/enhancements are easier to perform than non-modularized expert systems applications — i.e. each partitioned rule 'cluster' can be independently verified and validated. This is a significant advantage over expert systems with a non-partitioned knowledge-base, and can ultimately lead to lower expert system maintenance/enhancement costs, and with better documentation.

AUTOMATED TEST CONDUCTOR (ATC) DEMO

The preparation and launch of a contemporary space vehicle is a labor-intensive process caused by the need for cooperation between many interdependent systems. In the last decade commercial software tools, capable of capturing the knowledge and practical experience of an expert as well as represent the design of the system, have been developed that can contributed to saving both cost and schedule. However, most of todays intelligent program packages suffer from the inability to communicate or act cooperatively with conventional systems or similar systems, or operate in the R/T environment. Our previous R&D experiences have explored the use of expert systems technology in practical situations. We have demonstrated expert systems operate in near real-time, work cooperatively with conventional R/T systems, work interactively with object-oriented graphics, and integrate with an off-the-shelf relational database management system. These R&D projects demonstrated the essential capabilities needed for cooperation between distinct intelligent modules, real-time communication, and intelligent access to stored data files.

Based upon the Vehicle/Ground Intelligent Automation concept from Phase-I, we are attempting to demonstrate cooperation between multiple expert systems (and conventional software modules). This 'Automated Test Conductor' (ATC) prototype utilizes a Knowledge-bus (K-bus) approach for intelligent information processing by use of virtual sensors and blackboards to solve complex problems. It incorporates distributed processing of real-time data and object-oriented techniques for command, configuration control, and auto-code generation. Figure—10 pictures the final goals of this demonstration. Ultimately the K-bus would link conventional launch processing software and a distributed collection of expert system modules of various types with the vehicle avionics having perhaps some on-board intelligence for integrated health monitoring. The benefits would be proof of the concept R/T operability, scaled cost comparisons, and a testbed for V&V.

Abacus Programming Corp. and the LinCom Corp. were participating subcontractors.

Development Philosophies — The ATC demonstration itself will be developed according to four general philosophies: autonomy, distributed control by modular expert systems, information transfer at a high level, and the use of intelligent databases.

System level philosophies:

CRITICAL RESPONSE ISSUES FOR R/T OPS
Expert Systems — ADP 2302

√	• **Saliences (prioritized rules)**
	• **Priority Scheduling (dynamic saliences)**
	• **Progressive Deepening**
	• **Variable Precision Logic (depth of analysis wrt value of results = f(time))**
	• **Decision Analytic Techniques (depth of search wrt value of results = f(time))**
	• **R/T A* Search (best-first search)**
√	• **Bypass ES with Interrupt**
√	• **Interrupt Inserted Facts**
√	• **Single Valued Facts (2 fields vs 4 fields: (obj,val))**
√	• **Retract Seldom Used Facts**
√	• **Clause Ordering (RETE algorithm)**
√	• **Shorten Variable Names**
toolset	• **V&V Trimming (remove redundant facts, subsumption)**
√	• **Data Preparation (scaling, thresholding, changes-only in data)**
	• **Partitioning of the KB**
√	• **Faster Algorithms (OPS-83, PIE)**
√	• **Perform Math Functions Outside ES**
	• **Use Alternate Paradigm (frame, model-based, etc.. as suits problem)**
√	• **Quadruples Facts (4 fields: (obj,attrib.,value,confidence))**

Figure—9, Critical Response issues considered for R/T operations

• Autonomy — The ALS environment will require that support systems be relatively autonomous and capable of independent decision-making. This will reduce the need for a standing army of engineers and ease the impact of the anticipated loss of older experienced personnel.

• Distributed Control — The test conductor will coordinate distributed controls for ground systems equipment (GSE) and vehicle avionics systems. The GSE is a distributed system, distributed both functionally and physically. Functions such as test procedures, fault diagnosis, or scheduling may be performed by separate expert systems. Physical distribution will be permitted, with the demonstration performing on distinct workstations .

• High-level Intelligent Information Transfers — The demo will use symbolic or language links which will allow the modules to share common knowledge, while performing their distinct functions. This sharing will be intelligent in that it will anticipate knowledge requirements by the modules, and supply them in compatible forms. It is anticipated that the real-world system will supply this information through common real-time or non-real-time lines.

• Intelligent Databases — Data records and files will be handled according to specific heuristics which take into account interconnections and dependencies. Smart schematics, for instance, would cascade externally-caused changes to all relevant data records.

• Hardware Independence — The program will be developed so as not to be limited to a particular hardware network or machine. It is anticipated that the applications will be written in different languages, on different machines, using different operating systems or inference engines. The final system hardware is unknown; therefore, the test conductor will accommodate these varying components, with minimal alteration.

• Real-time Performance — The software selected must satisfy real-time performance requirements, when required. These requirements will be driven by the necessary integration to hardware and real-time software systems, such as the avionics software on the vehicle.

Software philosophies:

• Language Standards — Standard high-level languages shall be employed, to simplify development and maintenance. The language of choice is ADA. Other languages will be accepted, only when driven by higher standards, i.e. shells, cots packages, etc.

• UNIX Commonality — The R/T UNIX operating system will be the system of choice for the development and functioning environment of the ATC. This will provide for distributed system commonality while supporting real-time. UNIX is a wide-spread workstation operating system, and provides many libraries and abilities which are necessary for the ATC, such as interprocess communication, etc.

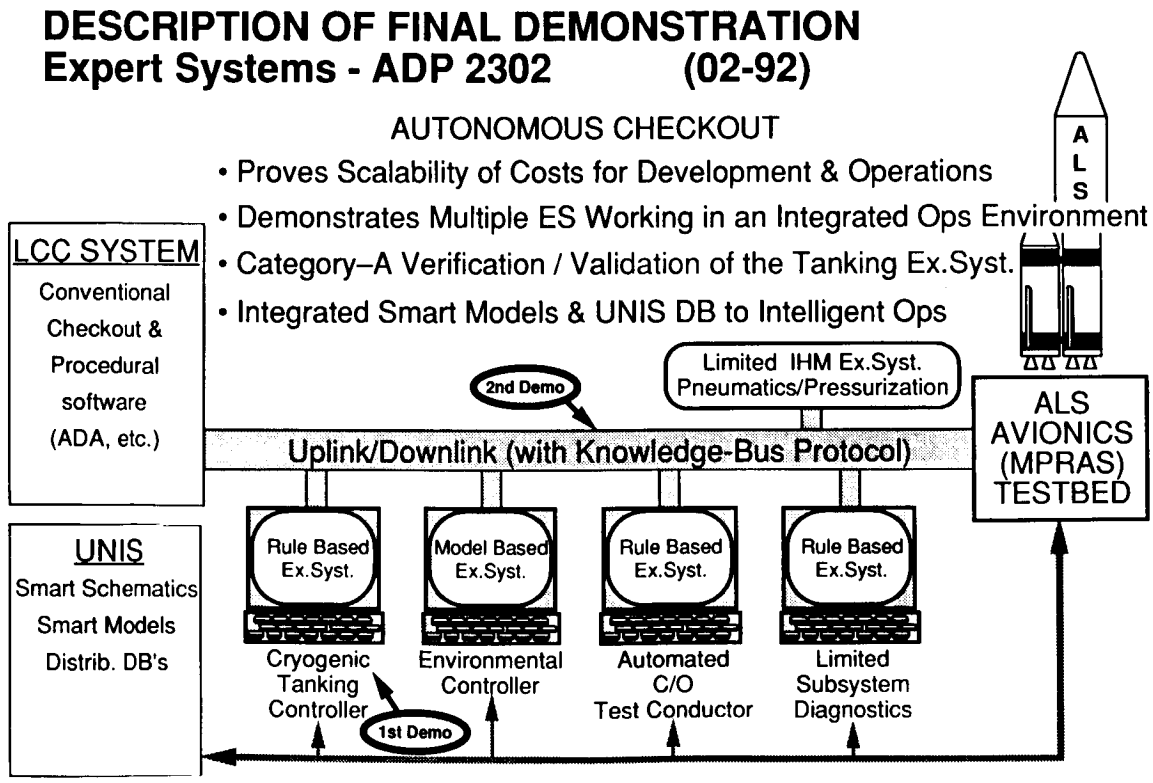
• Existing Toolkits — Commercial off-the-shelf (cots) products will be employed when they will adequately satisfy the task requirements. The benefits here are obvious.

• Object-Oriented Programming — The test conductor will be developed in accordance with object-oriented programming techniques. Currently the C++ language is targeted, promoting software reusability and functional modularity consistent with the object-oriented philosophy. TAE, a graphics package developed in C++ by NASA/Goddard, simplifies the use of XWindows, and promotes an iconic object-oriented user-interface.

• Shells and Environments — Commercial shells and environments that are proven to adequately satisfy the task requirements will be utilized. For the artificial intelligence portions of the testbed, such as the K-bus, CLIPS will be the shell of choice. CLIPS interfaces well with the C language and provides enough functionality to satisfy our R/T requirements. Other AI paradgms will be used as required.

• Permanent Data Storage — The ATC will need an imbedded database system, both for its system functions and its nodes. The database currently targeted is Oracle's relational database management system, which interfaces easily with the C language and is hosted on several workstations. A SQL-bridge will link the two.

Knowledge Bus — A critical component of the ATC demonstration is the Knowledge Bus. As its name implies, it provides a communication path for a higher level of information transfer, not simply data. The K-bus is a layered architecture in an object paradigm for development, integration and verification of distributed real-time systems, see Figure—11.



Figure—10, Autonomous vehicle checkout is possible using the K-bus approach

Systems implemented under the K-bus can include both knowledge-based and conventional procedural components. The K-bus features tools to ease development and coordinate functions that permit diverse applications to operate as a coherent system. Just as an operating system manages physical resources, the K-bus provides the means to access common reasoning services for embedded knowledge-based applications and analogous high-level services for procedural applications and V&V.

The K-bus concept was originally developed in Phase-I with a Design Specification for supporting a distributed network of cooperating expert systems serving an integrated Vehicle/Ground Mission Management System. It was anticipated that such a system would take the maximum advantage of semi-autonomous agent processes with knowledge-based communication and control to perform operations and vehicle/ground checkout. Phase-II applys these concepts to a working prototype:

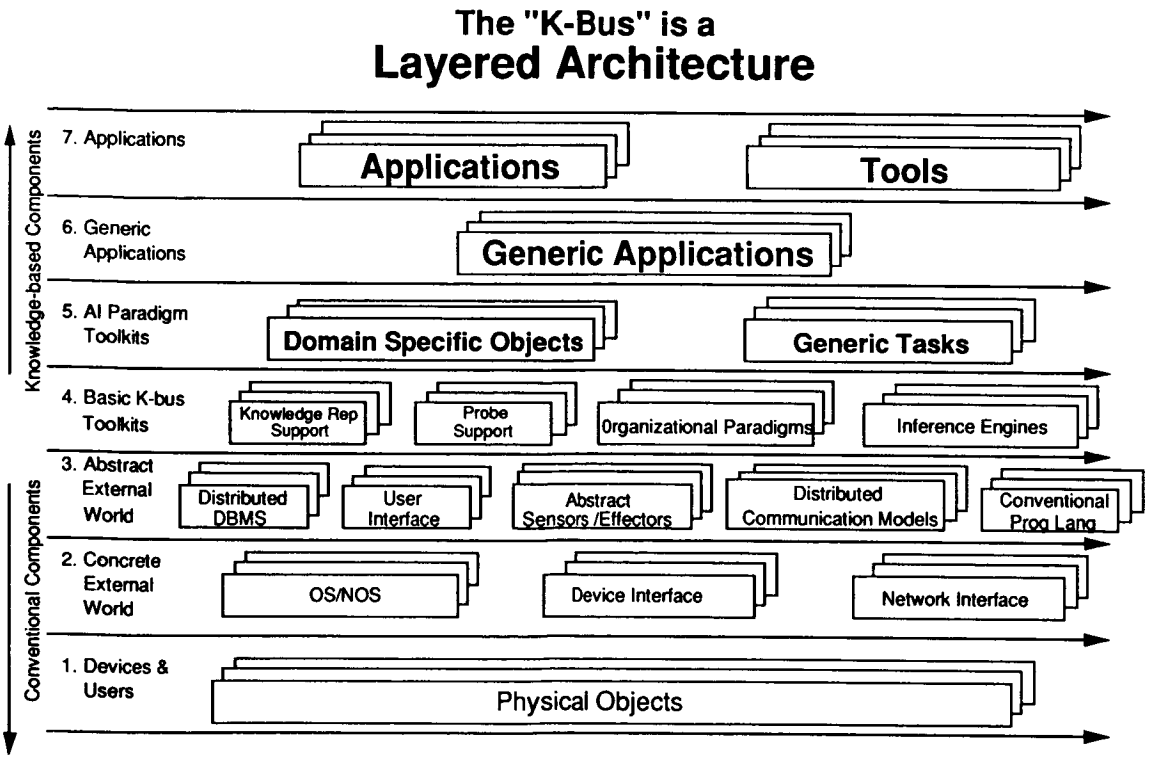
- Development of design specifications of the K-bus, and a User Manual describing how to use its facilities in developing a distributed application.
- Detailed design of the following K-bus objects: Object, SaveableObject, SystemCall, List, Buffer, String, Attribute, Socket, PostOffice, Finder, Message, KnowledgeUnit, KnowledgeSource, MessageManager, Agent.
- Implementation of these objects as a C++ library and Alpha testing of them with a simple driver program. This library was developed with Oasys C++ compiler and the Apple AU/X operating system.
- Rehosting the library from the development Oasys system to the Gnu C++ compiler and SUN workstation for integrated testing.

Aspects of the K-bus — As previously stated, the K-bus follows the distributed object oriented model of interaction between software modules, defined here to be loosely-coupled 'agents'. Further, this supports the open, continuous processing characteristic of cooperative systems, and which makes them much more complex than traditional consultation-based programs. This event-driven programming methodology is also shared by several conventional systems, such as X Windows. In each case, procedures ("event handlers") are associated with events that can occur asynchronously, such as the user clicking a mouse, or a database update.

Agent — The agent is the fundamental active entity in the K-bus, encapsulated as an object which communicates by messages. Currently an agent and its message manager occupy a Unix process, so its boundary exists not only as a software object but is also enforced at the operating system level. An agent is defined as a collection of knowledge sources and an organization. These knowledge sources may be implemented as expert systems or a conventional system. Each knowledge source has a list of capabilities and interests. This list matches questions it can answer and information it would like to be told. The agent advertises these attributes with the Finder and keeps a cache of other agents' capabilities and interests for subsequent communication.

An agent's specification thus permits implementation along several sizes of granularity. Internally, it can be a whole organization of problem solvers, or just a simple C program. It has a scheduler component for control of its knowledge sources and is not necessarily serial. Its state may be dormant or active, but currently most agents are eternally vigilant or waiting for a reply. For efficiency reasons in Unix-like environments a large grain may be preferred, and this can be used at the next layer up as a generic task (an agent which is a specialist in one area of problem solving).

An agent's capabilities and interests represent a model of its goals, plans, abilities and needs that other agents can use for



Figure—11, Autonomous vehicle checkout is possible using the K-bus approach

cooperation. An agent can choose not to cooperate by not advertising this model, but in general they can build up more extensive models of each other by starting with the originally advertised capabilities and interests and then learning from experience by caching results. For example, two agents may have a capability to do arithmetic, but by trying each the faster one is identified and will be preferred in future requests. An agent can have a reflective ability by installing probes in itself (for example, to measure the number of rules fired by a knowledge source's inference engine). This ability allows it to monitor its progress and interrupt if necessary. The combination of agents into a cohesive problem-solving team is achieved by creating an organization. Figure—12 is an example of the internal organization of a complex agent.

Post Office – Each agent has a Post Office object, which queues incoming messages and permits addressing by name, rather than location. The Post Office uses a distributed Finder object, which keeps track of the addresses of active objects and maps them to their globally unique names. Furthermore, agents can advertise certain attributes (see later section) which are also registered with the Finder and permit communication by semantics rather than just syntactic names.

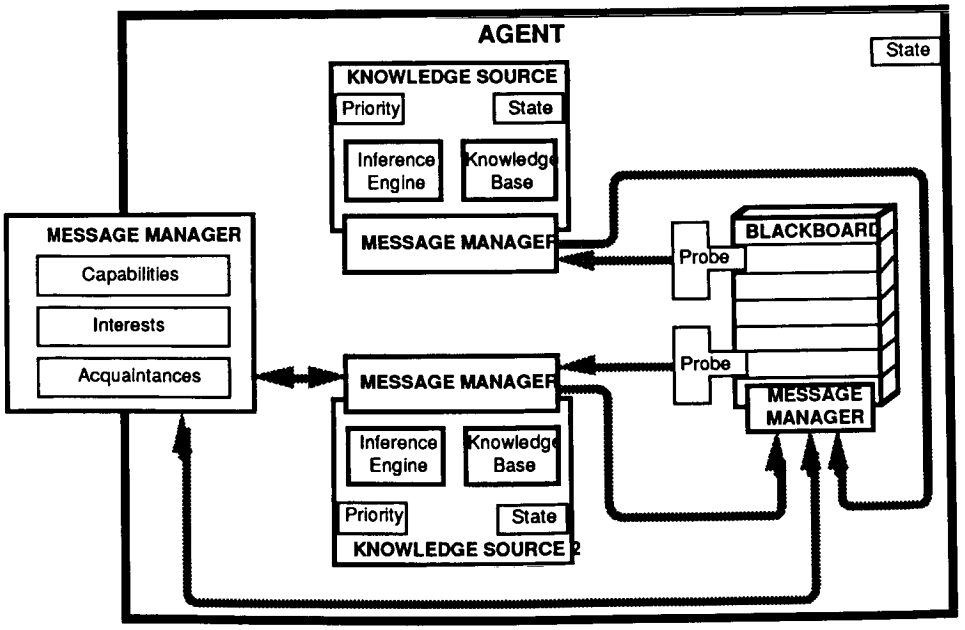
Message — The interaction medium is the message, the glue which enables the transfer of data and control between the agents. A message contains fields which identify the sender and receiver, an object (such as a question or answer) an optional time tag and list of attributes (which may include its expiration date or application-specific information). Control is passed by messages which represent remote procedure calls - they are intercepted by an agent's message manager. The message manager is responsible for converting messages to procedures and keeps a queue of questions received together with their askers (for subsequent direction of replies). Remote procedure calls by default are asynchronous (the caller doesn't block and wait for its completion), but may be synchronous if required (easier to program as it fits the conventional procedural language model). The question of whether the receiving agent blocks until it processes the request depends on the organization used: if the agent does - it is under the control of the sender (a client-server relationship); if not - it is autonomous. Of course, requests to lower-level services (such as a database manager) are processed synchronously - only high-level agents can own a thread of control.

Organization — An organization is simply a collection of agents who know each others' capabilities and interests, this is an implicit specification, encapsulated by knowledge existing in each agent. In contrast to structural definitions of organizations, this model is adaptive, since agents can compute who knows how to answer a question. New relationships can form within the organization. One agent can be programmed to act as a manager, delegating work to other agents according to their advertised capabilities, monitors their progress using probes and adjusts their position in the organization accordingly.

A method to combine agents more indirectly is by sharing access to a blackboard.

Blackboard — A blackboard is realized in the K-bus as a restricted subclass of agent - it is a passive server which is interested in everything (or at least whatever it is programmed for). Agents post information on the blackboard by sending it messages, they install probes on it to gather information resulting from matching events plus several current and historical conditions. A blackboard is thus a semi-permanent communication space, but also acts as a mechanism for a loosely-coupled organization whereby several agents can combine partial results without repeated inter-agent communication. It is more than a global database, in that the probes' histories provide a short-term memory and record of partial matches, so that new additions and requests can be processed quickly (in the style of the Rete algorithm for rule-based systems); in contrast, database queries are processed one at a time. This is an object-oriented version of the blackboard concept, and it is important to contrast it with blackboard systems which contain a centralized scheduler in control of the serial execution of agents - in the K-bus the agents are autonomous, and questions of parallelism and interference are answered by the message-passing architecture.

The blackboard's internal structure may be partitioned, to allow a for hierarchy of spaces available to groups of agents, but the external interface is ignorant of the internal structure of objects posted on it. Although logically centralized, it may be



Figure—12, An example of a complex Agent composed of several objects

physically distributed for performance reasons - in this case, consistency must be maintained using techniques (e.g. multiple copies, deadlock avoidance) borrowed from distributed databases. A blackboard is demonstrated in Figure—13.

Interim Demonstration Configuration — The present ATC prototype was developed as an early assessment of the difficulties involved in operating a network of cooperating expert systems. The demo system is comprised of five expert systems (ES). Primary control is represented by the Automated Test Conductor (ATC) user interface. A message router, equivalent to the Finder mentioned above, handles information flow and command/control. The interface to the ATC is via an oop iconic display window, much like a Macintosh. This interim demo emphasized the concepts of distributed control and oop communication. Via a UNIS-like data management interface, oop scripts can be developed for a test scenario. Process ES objects may be assigned to any workstation or mainframe in the network and given initialization information. All results flow back to the ACT operator window. Each of the distributed sub-processes open their own respective windows on their hosted machine for inspection. This entire process is stored in object form and when initiated the software is auto-code generated, distributed, and executed. Figure—14 shows the five expert system configuration. The ATC user interface is in Figure—15.

REFERENCES

[1] Szatkowski, Dr. G., 'KB Partitioning Design for CLIPS', CLIPS Users Conference, Houston Tx., 1990.

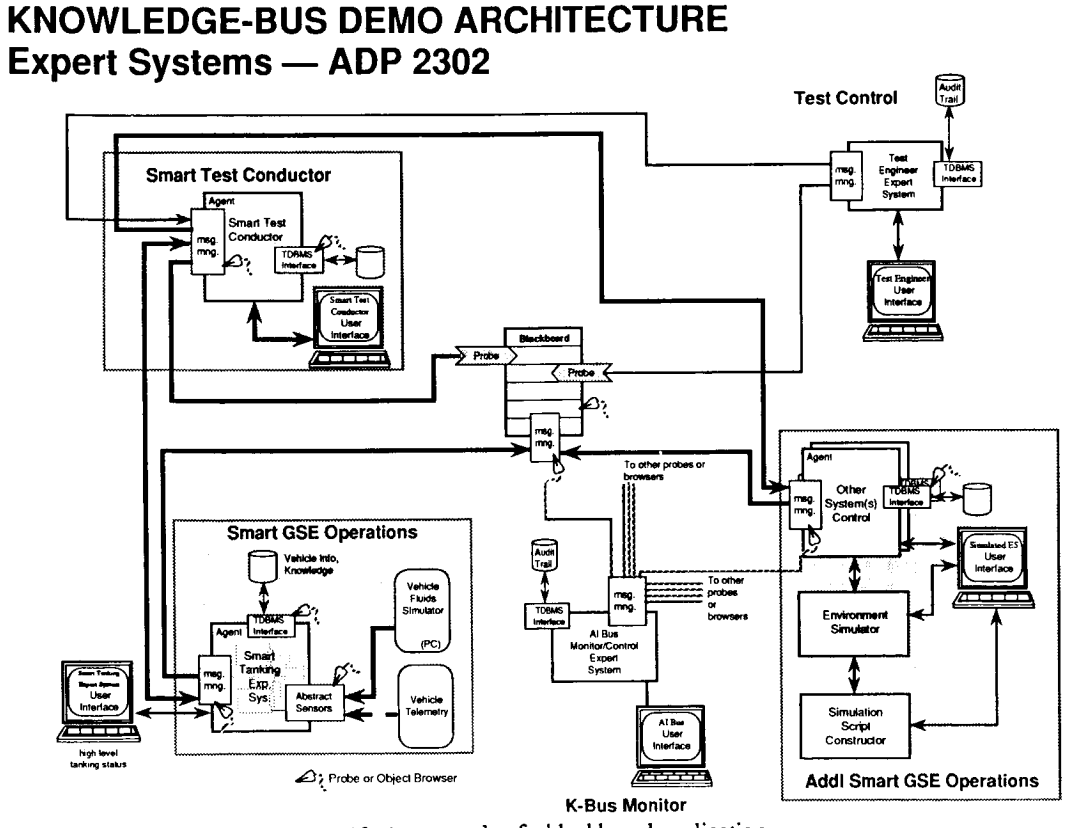
[2] Szatkowski, Dr. G., 'AI Decision Support for Low Cost Launch Vehicle Integrated Mission Operations', SOAR, Dayton Oh., 1988.

[3] Szatkowski, Dr. G., 'ALS STRESS Final Report', USAF contract F33615-87-C-3620, 1990.

[4] Szatkowski, Dr. G., 'Approaches to V&V of Imbedded Decision Support Systems Applied to Launch Vehicle Ops', Validation and Testing KB Systems Workshop, IJCAI, 1989.

[5] Szatkowski, Dr. G., 'ALS Expert Systems ADP-2302 Final Report', USAF contract F304701-88-C-0110, 1990.

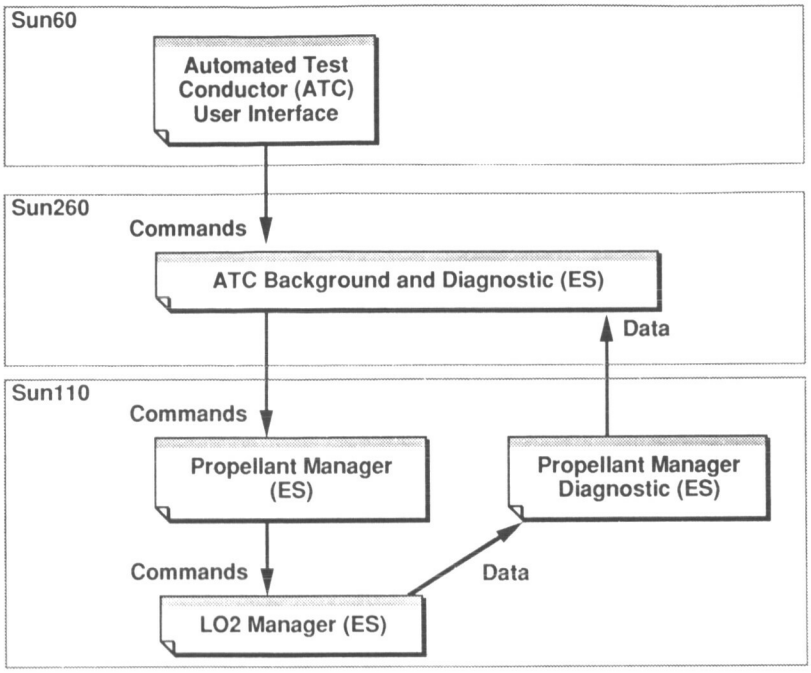
[6] Le T. and Homerier P.,'Portable Inference Engine, PIE', Aerospace Corporation, LA Ca.



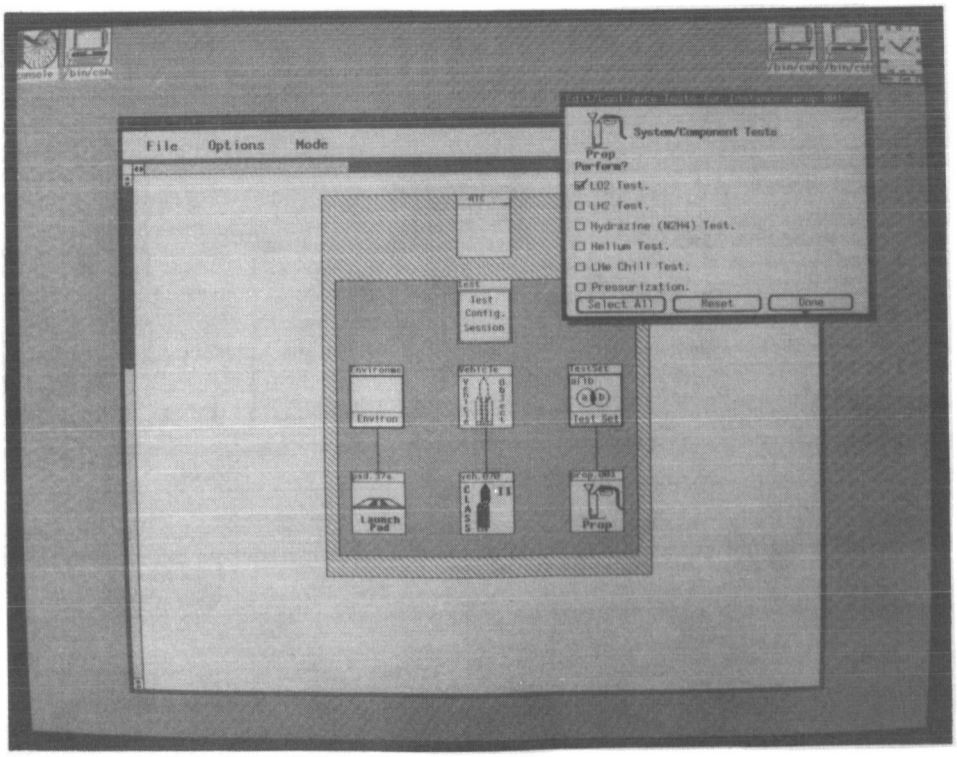
Figure—13, An example of a blackboard application

ADP 2302 - INTERIM DEMO
Vehicle Ground Intelligent Automation

Command, Control, and Data Flow



Figure—14, ATC prototype demonstrated distributed multiple-Expert System cooperation



Figure—15, Photograph of the ATC object-oriented Scripting capability